FFTs on BG/L

Status and Methods

Franz Franchetti

Institute for Applied Mathematics
and Numerical Analysis

Vienna University of Technology (TU Wien)

Acknowledgements

This work was supported by the Austrian Science Fund FWF in the context of Project 5 of the SFB AURORA (Ueberhuber).

Thanks to our partners at IBM and LLNL for their help and support.

People involved

Vienna University of Technology

F. Franchetti, S. Kral, J. Lorenz, C. W. Ueberhuber, P. Wurzinger

Carnegie Mellon University (SPIRAL)

M. Pueschel, Y. Voronenko

FFTW

M. Frigo

Outline

- Current Status
- FFTs on BG/L – the challenges
- Utilizing Double Hummer in FFT implementations
  - FFTW
  - SPIRAL
  - Vienna MAP vectorizer
- Summary and outlook

Outline

- **Current Status**
- **FFTs on BG/L – the challenges**
- **Utilizing Double Hummer in FFT implementations**
  - **FFTW**
  - **SPIRAL**
  - **Vienna MAP vectorizer**
- **Summary and outlook**

FFT Library for BG/L

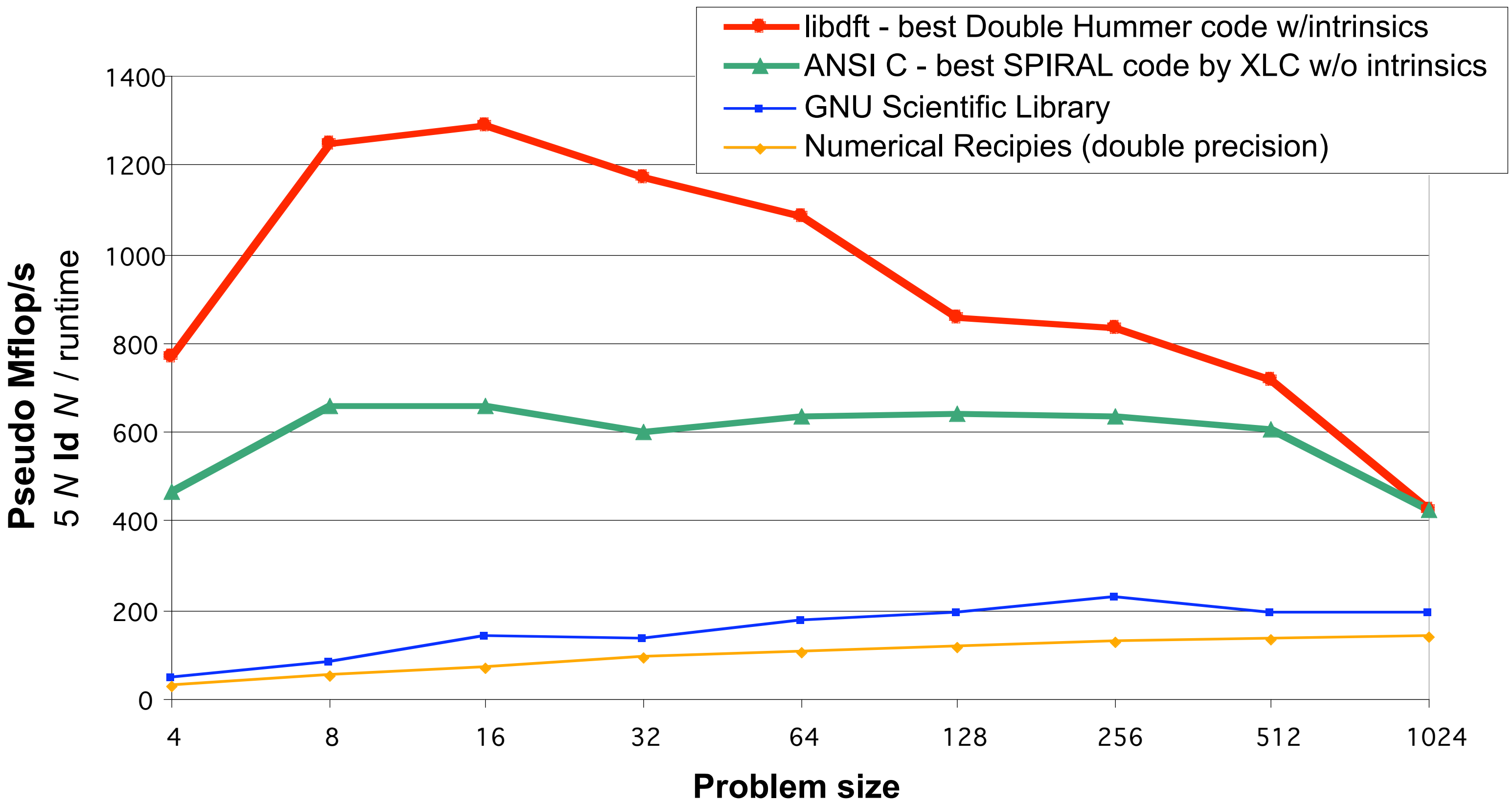
Current Status

- Very fast – fastest FFT on BG/L
- Optimized for Double Hummer
- Experimental, work in progress
- Currently targets *single processor*
- Automatically generated code (XLC intrinsics + C99)

Next steps

- Utilize knowledge gained by visit to IBM T. J. Watson Research Center
- Make available to LLNL users
- Utilize both processors on the PowerPC 440D

Measured Performance



DFT 2^n, double precision, complex to complex PowerPC 440D at 500 MHZ

Speed-up (to best ANSI C code) up to 2

Outline

- Current Status
- **FFTs on BG/L – the challenges**
- Utilizing Double Hummer in FFT implementations
  - FFTW
  - SPIRAL
  - Vienna MAP vectorizer
- Summary and outlook

State of the Art FFTs

$x \mapsto DFT_N x$ where $DFT_N = [\omega_N^{jk}]_{j,k=0,1,\ldots,N-1}$ with $\omega_N = e^{2\pi i/N}$

Discrete Fourier transform: **O(n^2) operations**

Fast Fourier transform: **O($n \log n$) operations**

Number of arithmetic operations is *not* strongly correlated with runtime

- Deep memory hierarchies
- Superscalar processors
- ISA extensions (FMA, SIMD, prefetching,…)

  …

Fast FFT implementations

- Vendor libraries
- Automatic performance tuning systems

Portable State-of-the-art DFT Implementations

Automatic Performance Tuning Systems

- FFTW: a library for FFTs
- SPIRAL: a library generator for DSP transforms

Characteristics of advanced DFT software

- Automatically generated and HW adapted libraries
- Large sections of straight-line
  single static assignment (SSA) code
  1000s of operations using 1000s of temporary variables
- Utilization of modern ISA extensions *required*
  fused multiply-add (FMA) instructions,
  short vector SIMD instructions (Double Hummer),…

How to get Double Hummer support?

Straight-line SSA Code

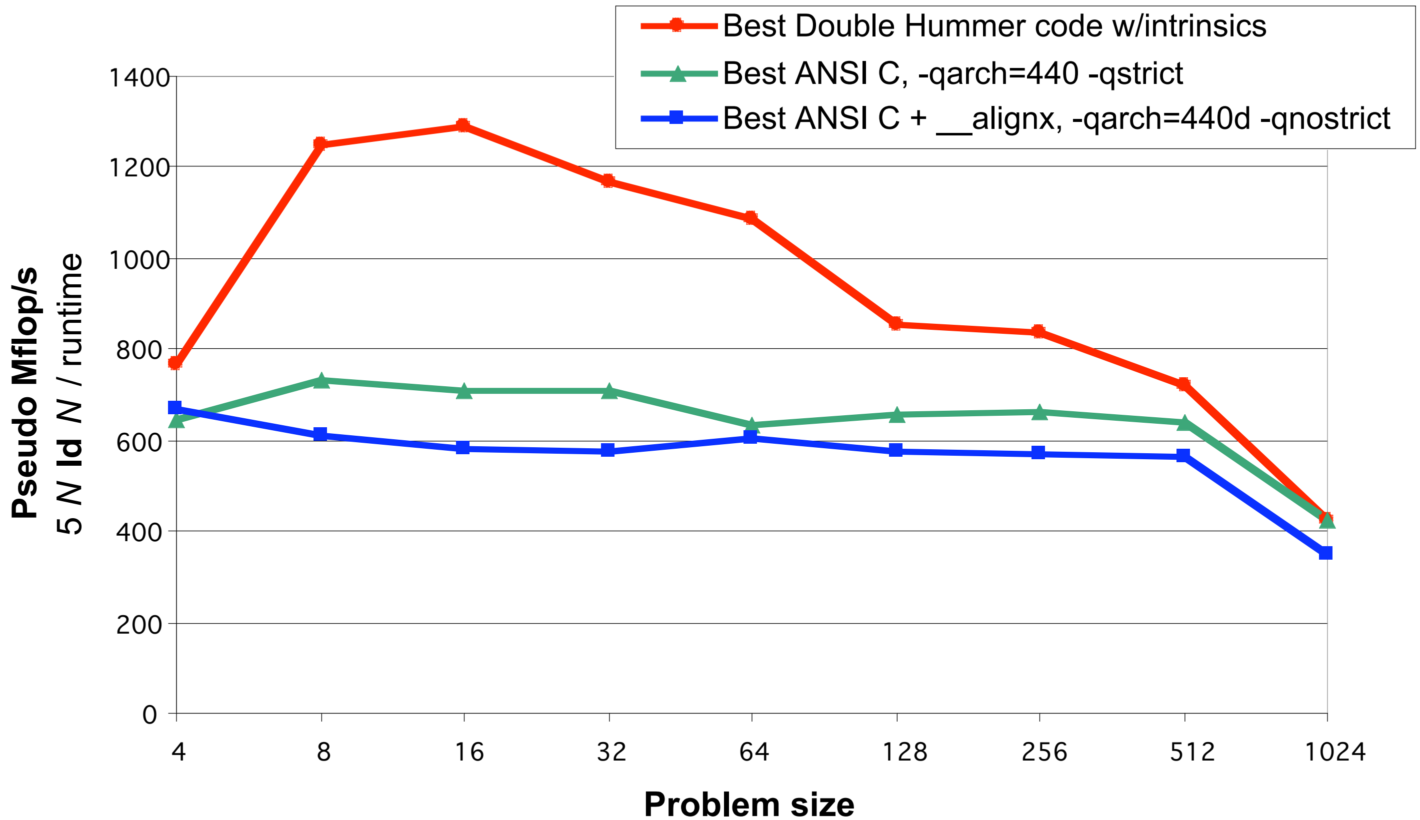
```
void DFT_64(double *y, double *x)
{
    __alignx(16,x);
    __alignx(16,y);
    double f0;
    dou    f1;
    ..
    d
    f0 =
    f1 = x[1
    f2 = x[0] +
    ...
    f7 = x[33] + x[97];
    f8 = f2 - f6;
    f9 = f3 - f7;
    ...
    f1196 = 0.2902846772544623 * f70
    f1197 = 0.9569403357322089 * f700;
    ...
    f1206 = f1186 + f1198;
    f1207 = f1187 - f1199;
    y[94] = f1202 - f1206;
    ...
    y[127] = f1201 - f1204;
    y[62] = f1200 - f1205;
    y[63] = f1201 + f1204;
}
```

Vectorization by XLC for BG/L possible?

SPIRAL generated DFT_{64}

- Straight-line SSA code
- Only binary Operations: **`add`**, **`sub`**, **`mul`**
- 1208 temp vars
- 1336 assignments

XLC Vectorization



DFT$_{2^n}$, double precision, complex to complex PowerPC 440D at 500 MHZ

XLC vectorization and FMA extraction can't accelerate our DFT codes

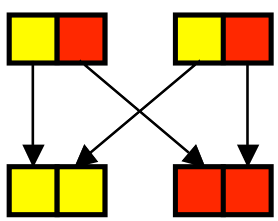
Double Hummer DFT Challenges

Fused multiply-add

- DFT is not localy balanced w.r.t. adds and muls
- FMA extraction changes data access locality

Double Hummer vectorization

- DFT is complex-to-complex transform, however
  real arithmetics optimization is applied by SPIRAL and FFTW
- Can vectorize codes by inserting
  `fmr`, **`fxmr`**, **`fsmfp`**, **`fsmtp`** instructions, however
  cost is prohibitive: 1 **`fxmr`** = 4 flops



440 FPU	**440D Double Hummer**
Variable renaming 0 instructions	Register operation 3 instructions (XLC) 12 flops wasted

Utilizing Double Hummer in State-of-the-art DFT Codes

FFTW 3.01

- Vectorization of complex-to-complex FFTs
- Folding all data reorganization into Double Hummer FMAs possible

SPIRAL-SIMD

- Vectorization of DSP transforms for *n*-way short vector machines
- Automatic vectorization on symbolic level

Vienna MAP vectorizer

- Two-way vectorizer for straight-line SSA codes
- Plug-in for SPIRAL, FFTW, ATLAS,…

Outline

- Current Status
- FFTs on BG/L – the challenges
- **Utilizing Double Hummer in FFT implementations**
  - **FFTW**
  - SPIRAL
  - Vienna MAP vectorizer
- Summary and outlook

FFTW

FFTW: Hardware adaptive FFT library

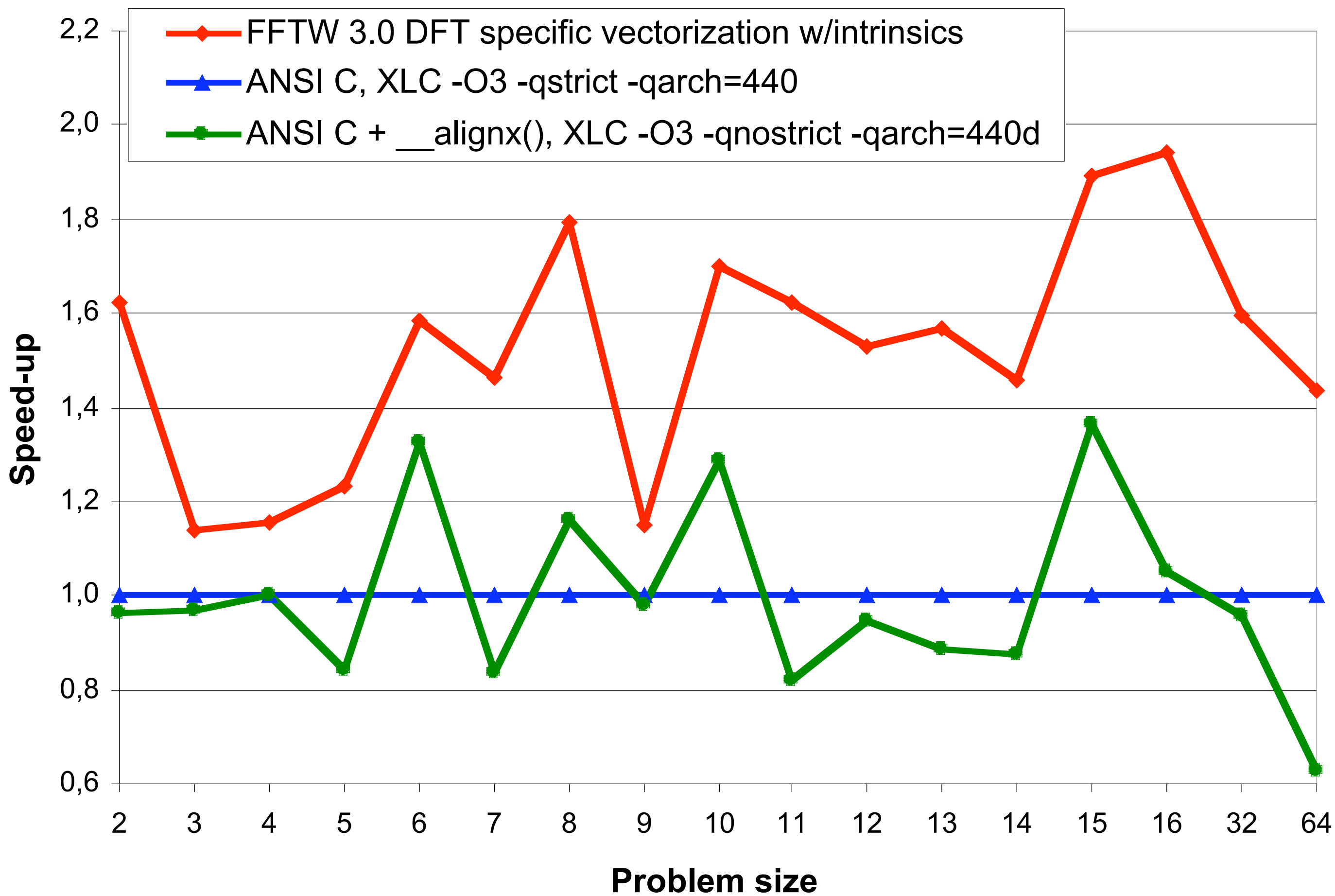
Matteo Frigo
Steven G. Johnson (MIT)
`www.fftw.org`

- Various basic routines (codelets) are combined to compute the desired FFT
- Codelets are generated automatically by codelet generator **`genfft`**
- Codelet combination is determined at runtime by dynamic programming

FFTW for BG/L

- Version 3.0 provides complex-to-complex FMA SIMD codelets well suited for BG/L
- Method depends on specific properties of complex DFTs
- FFTW 3.0 port to BG/L is underway (Franchetti and Frigo)

Performance of FFTW 3.0 SIMD Codelets



DFT$_N$, double precision, complex-to-complex PowerPC 440D at 500 MHZ

Outline

- Current Status
- FFTs on BG/L – the challenges
- **Utilizing Double Hummer in FFT implementations**
  - FFTW
  - **SPIRAL**
  - Vienna MAP vectorizer
- Summary and outlook

SPIRAL: A DSP Library Generator

José Moura (CMU)
Jeremy Johnson (Drexel)
Robert Johnson (MathStar)
David Padua (UIUC)
Markus Püschel (CMU)
Viktor Prasanna (USC)
Manuela Veloso (CMU)

SPIRAL

- Code generation for DSP transforms (DFT, DCT, ...)
- Automatic platform adaptation
  on algorithm and implementation level

Facts

- For a given transform there are many different algorithms
  (equal in arithmetic cost, different in data flow)
- The best algorithm and its implementation is platform dependent

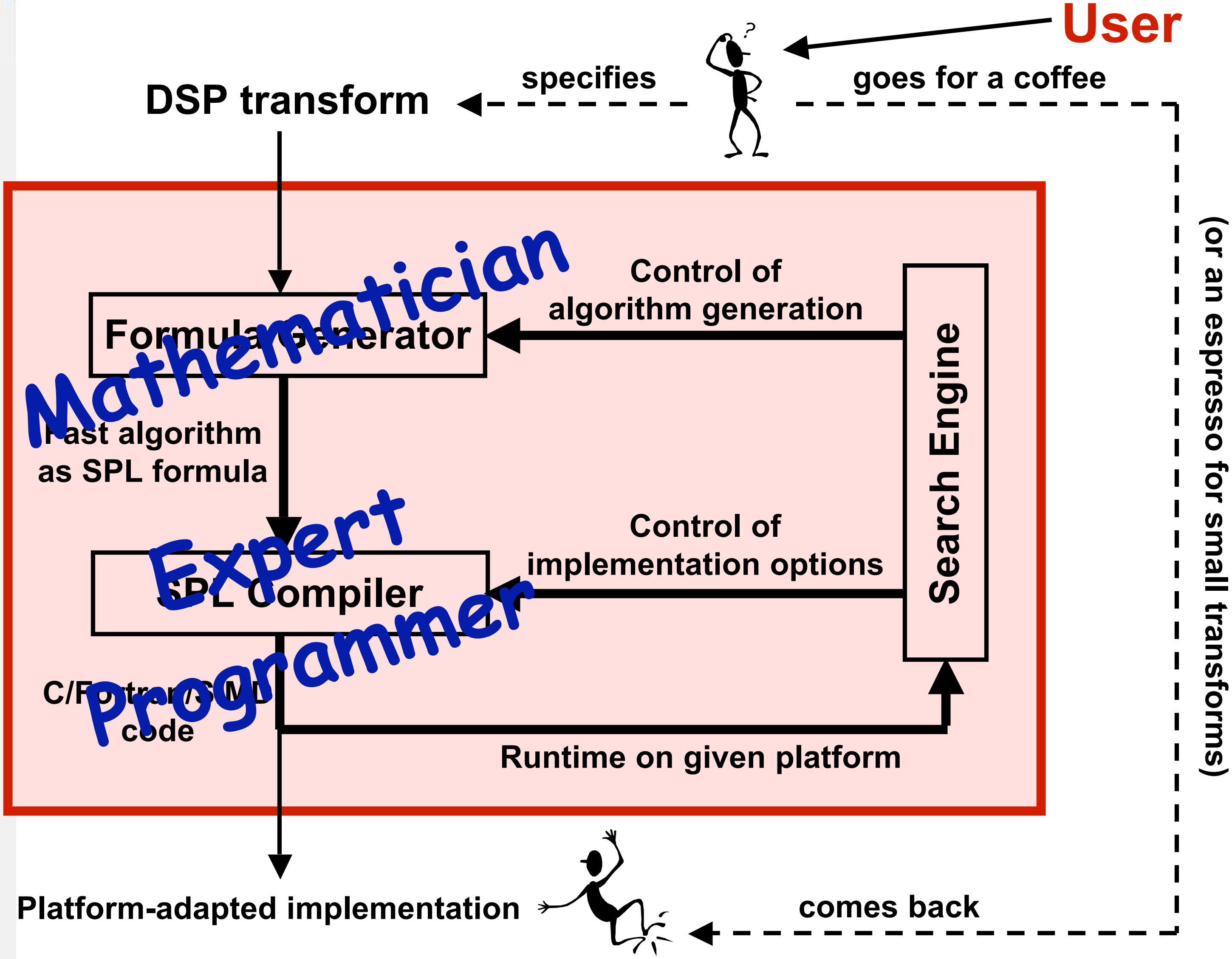
Approach

Automatic algorithm generation
\+ Automatic translation into code
\+ Intelligent search for "best version"

= Generated platform-adapted implementation

`www.spiral.net`

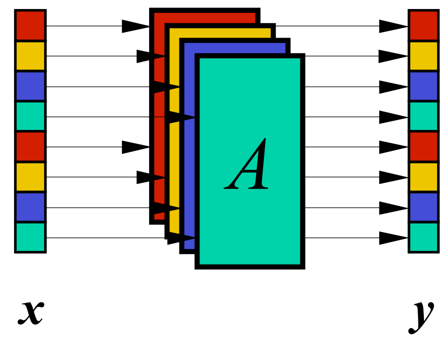
SPIRAL System



Using SPIRAL to Generate Double Hummer Code

$$y := (A \quad I_4)x$$

vector length



naturally represents vector operation

- Use macro layer (Portable SIMD API) to hide Double Hummer specifics
- Vector code generation in two steps
  1. Symbolic vectorization (Extend Formula Generator)
  2. Code generation (Extend SPL Compiler)
- SPIRAL-SIMD is ported to BG/L and produced optimized Double Hummer code with real hardware in optimization loop
- Experimental FMA extraction
- Core of BG/L DFT library

F. Franchetti, M. Püschel: „A SIMD Vectorizing Compiler for Digital Signal Processing Algorithms“, In Proceedings of IPDPS 2002

F. Franchetti, M. Püschel: „Short Vector Code Generation for the DFT“, In Proceedings of IPDPS 2003

Outline

- Current Status
- FFTs on BG/L – the challenges
- **Utilizing Double Hummer in FFT implementations**
  - FFTW
  - SPIRAL
  - **Vienna MAP vectorizer**
- Summary and outlook

The Vienna MAP Vectorizer

Industry-standard automatic vectorization for straight-line code is *insufficient*

- Vectorization for short vectors (length 2, 4,...) is not possible in a straightforward manner
- Most Industry-standard vectorizing compilers are loop based, these kernels codes are straight-line code
- XLC for BG/L vectorizes straight-line code, however cannot vectorize DFT straight-line code well

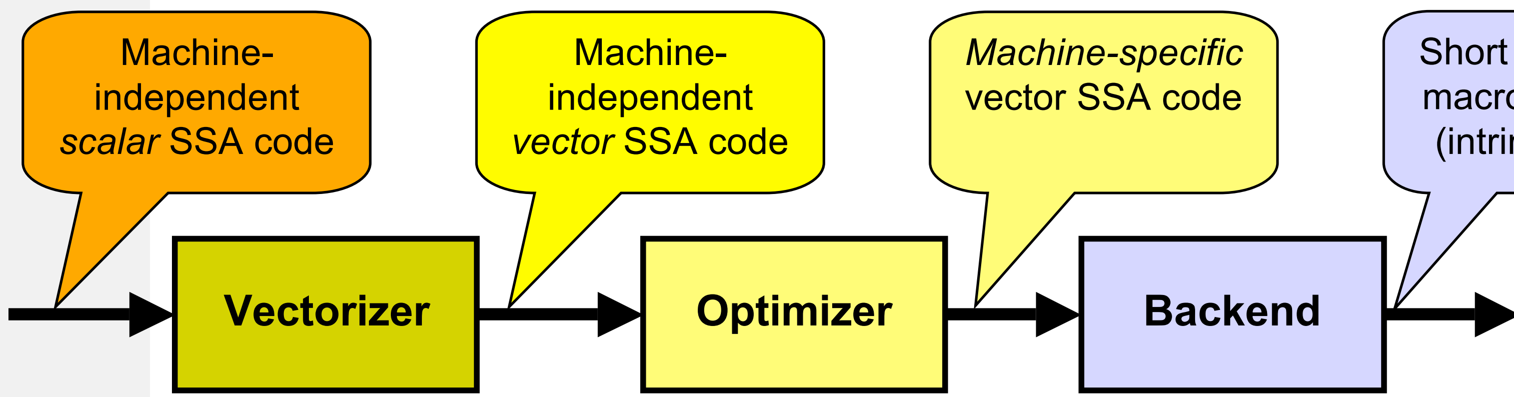
The Vienna MAP vectorizer targets two-way vector units like IBM's Double Hummer, Intel's SSE2, and AMD's 3DNow!

Goal: Automatic source-to-source vectorization for high-performance numerical straight-line SSA code

MAP Vectorizer Overview

Source-to-source vectorization

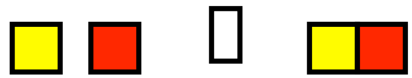
- Special-purpose vectorizing compiler
- **Input:** directed acyclic graph (DAG) of numerical SSA code e.g., generated by FFTW's codelet generator **`genfft`,** SPIRAL's SPL compiler, ATLAS,...
- **Output:** Vector code utilizing macros (intrinsics)



S. Kral, F. Franchetti, J. Lorenz, C. W. Ueberhuber: „SIMD Vectorization of Straight Line Code", In Proceedings of EuroPar 2003

Vectorization Concept

Scalar temp vars ▯ *Tupels* of scalar temp vars



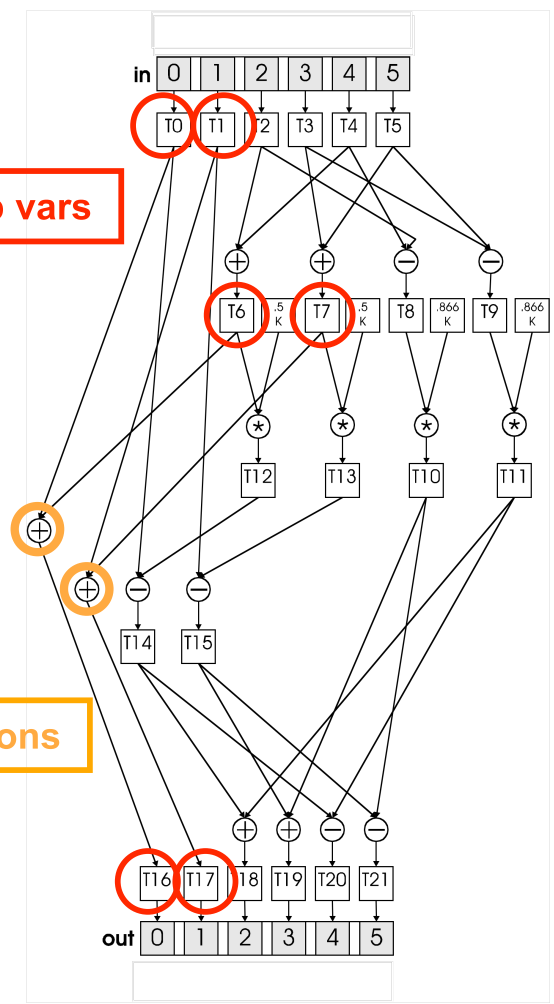
- **Variable fusion**
  - Temporary variable = scalar variable
  - Tupels of temporary variables = SIMD vector variables
- **Vectorization**
  Obtain a vector DAG operating on tupels
  - Any temp var is included in exactly one tupel
  - Vector DAG must be compatible to SIMD instructions
- **Operation fusion**
  Vector DAG implies SIMD operations

Example: DFT_3

Scalar DAG

Scalar temp vars

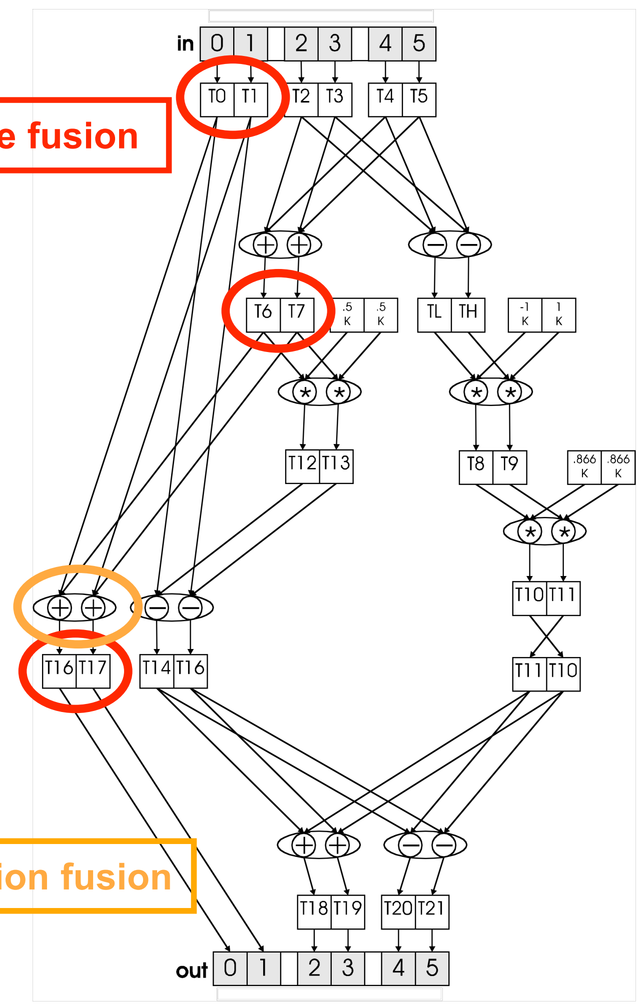
Scalar operations



Vector DAG

Variable fusion

Operation fusion



Implementation Details

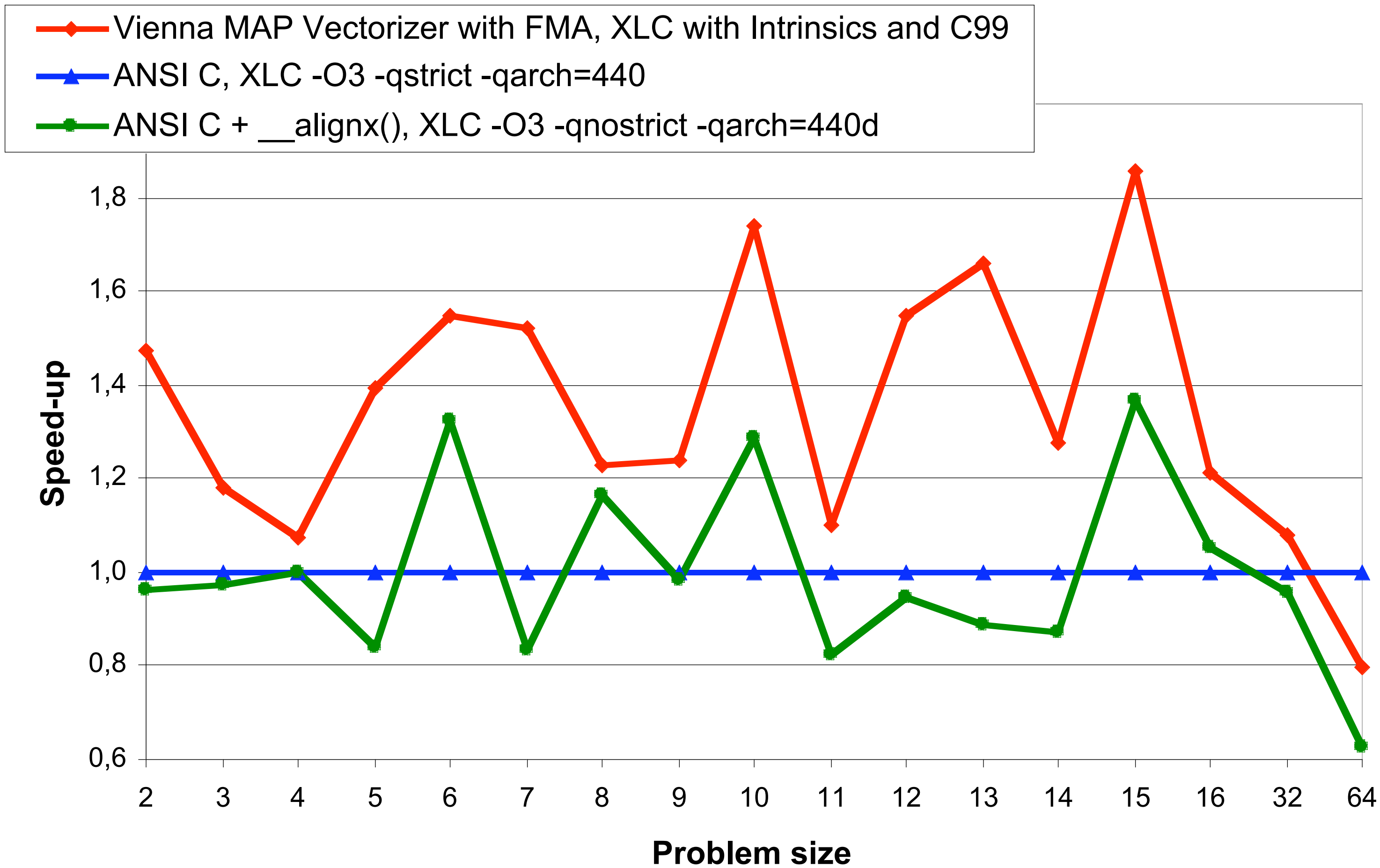
Special search engine

- Depth-first search with chronological backtracking
  on DAG corresponding to SSA code
- Implemented in OCaml (functional language)
- Reasonable runtime: <1s for 2,000 statement SSA code
  Huge search space but many possible solutions

Heuristics

- Restricted set of vector instructions to prune search space
- Search the DAG from stores towards loads
- Different vectorization levels
  If required, resort to suboptimal solution

MAP Performance for FFTW Codelets



DFT_N, double precision, complex-to-complex PowerPC 440D at 500 MHZ

MAP Vectorized Code

MAP Vectorized DFT_{64}

- XLC intrinsics: memory access and arithmetic operations
- C99 complex syntax for cross moves

```
static const _Complex double __align(16) VECT_CONST1 =
          __cmplx(-1.000000000000000, -1.000000000000000);
...
static const _Complex double __align(16) VECT_CONST21 =
          __cmplx(+0.634393284163645, +0.773010453362737);

void DFT_64(double *y, double *x)
{
   _Complex double f0;
   ...
   _Complex double f603;
   f0 = __lfpd((double *)(x+64));
   f1 = __lfpd((double *)(x+0));
   f2 = __fpadd(f0,f1);
   f3 = __fpmadd(f0,VECT_CONST1,f1);
   ...
   f417 = __cmplx(__creal(f415),__creal(f416));
   f418 = __cmplx(__cimag(f415),__cimag(f416));
   ...
   f602 = __fpmadd(f511,VECT_CONST2,f407);
   f603 = __fpmadd(f358,VECT_CONST3,f476);
   __stfpd((double *)(y+34), t602);
   __stfpd((double *)(y+98), t603);
}
```

Outline

- Current Status
- FFTs on BG/L – the challenges
- Utilizing Double Hummer in FFT implementations
  - FFTW
  - SPIRAL
  - Vienna MAP vectorizer
- **Summary and outlook**

Summary

Results and ongoing development

- **Experimental DFT library for BG/L is already very fast**
- **FFTW 3.01 for BG/L**
  First codelet runtime results utilizing Double Hummer
- **SPIRAL-SIMD/BGL**
  Double Hummer DFT implementation for $N=2^1$ to $N=2^{16}$
- **Vienna MAP vectorizer**
  Supports Double Hummer,
  Connected to FFTW and SPIRAL for BG/L

Medium-term goals

- Utilization of *both* CPUs of PowerPC 440d in FFT kernels in *computation offload mode*
- MPI parallel version

We are very interested to run for Gordon Bell Award with FFT-intensive LLNL application on BG/L